活动选择问题

与算法导论(第三版)章节的对应关系

- Chapter 16.1. 活动选择问题
- 涉及知识点包括：
  - 活动选择问题的描述
  - 活动选择问题的动态规划算法
  - 活动选择问题的贪心算法

活动选择问题

- 多个活动竞争资源的调度问题：尽可能多地选择互不冲突的活动
- 设有n个活动(activity) $S = \{a_1, a_2, \ldots, a_n\}$，均要使用某资源(如教室)，该资源使用方式为独占式，一次只供一个活动使用。
  - 每个活动a_i发生的时间为$[s_i, f_i), 0 \leq s_i < f_i < \infty$
  - 两活动a_i, a_j相容(compatible不冲突)是指：$[s_i, f_i), [s_j, f_j)$不重叠，满足$s_i \geq f_j \ or \ s_j \geq f_i$。即：一活动的开始时间大于等于另一活动的完成时间。
  - 活动选择问题：选择最多的互不冲突的活动，使相容活动集合最大。即：$A \subseteq S$，A中活动互不冲突且$|A|$最大。

活动选择问题

- 例: (*按完成时间递增序排列)

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	8	9	10	11	12	13	14

  - 问题的解：$A1 = \{a_3, a_9, a_{11}\}, A2 = \{a_1, a_4, a_8, a_{11}\}, A3 = \{a_2, a_4, a_9, a_{11}\}$
  - 最优解： $A2$和$A3$

- 此问题可用迭代方法直接给出贪心算法，但为比较和动态规划之关系，下面先考虑动态规划解法。

活动选择问题

活动选择问题的最优子结构

- 子问题空间可描述为：$S_{ij} = \{a_k \in S: f_i \leq s_k < f_k \leq s_j\}$

  S_{ij}是S的子集，它包含那些(在活动a_i完成之后开始)and(可在活动a_j开始之前完成)的活动a_k。

  亦即：S_{ij}中所有活动a_k与活动a_i、a_j相容，不妨称a_i、a_j和子集S_{ij}相容，因此a_k也与所有不迟于f_i完成的活动以及与所有不早于s_j开始的活动相容。

  注意：S_{ij}中可能有冲突的活动。

  为方便处理，设有两个虚拟的活动a_0和a_{n+1}，并且假定$f_0 = 0, s_{n+1} = \infty$

  因此，$S = S_{0,n+1}$，$0 \leq i, j \leq n+1$

活动选择问题

- 为了更严格定义i和j的范围，假定所有活动已按完成时间的单调增有序进行排序：

$$f_0 < f_1 \leq f_2 \leq \cdots \leq f_n < f_{n+1}$$

∵ $S_{ij} \neq \emptyset$, 若$i \geq j$ //易于用反证法证明

∴ 只考虑S_{ij}, 当$i < j$。因此i, j的范围是$0 \leq i < j \leq n+1$

活动选择问题

- 如何分解问题？
  - 设子问题 $S_{ij} \neq \emptyset, a_k \in S_{ij}, f_i \leq s_k < f_k \leq s_j$,若S_{ij}的解中选择了a_k，使用a_k产生S_{ij}的两个子问题：

    S_{ik}: 包括在a_i完成之后开始，在a_k开始前完成的那些活动

    S_{kj}: 包括在a_k完成之后开始，在a_j开始前完成的那些活动

    均是S_{kj}的子集，这两子集与a_k相容
  - S_{ij}的解显然是S_{ik}和S_{kj}解的并，再加上a_k。
- 注意S_{ik}和S_{kj}已去掉了那些与a_k冲突的活动，而这些活动原来可能在S_{ij}中。

活动选择问题

- **最优子结构**

  设S_{ij}的最优解是A_{ij}，$a_k \in A_{ij}$，

  则两子问题S_{ik}和S_{kj}的解A_{ik}和A_{kj}也必须是最优的

  易用反证法证明：

  A_{ik}和A_{kj}是**独立的，可用cut-and-paste证明**

- **用子问题的最优解构造原问题的最优解**

  $$A_{ij} = A_{ik} \cup \{a_k\} \cup A_{kj}$$

  整个问题的最优解是$S_{0,n+1}$**的一个最优解**

活动选择问题

一个递归解

设$C[i,j]$表示S_{ij}的**最优解的值**，即S_{ij}中**相容活动最大子集的size**

$C[i,j] = |A_{ij}|$

1) 当$S_{ij} = \emptyset, C[i,j] = 0, i \geq j$ // $\because A_{ij} \subseteq S_{ij}, \therefore A_{ij} = \emptyset$

2) 当$S_{ij} \neq \emptyset$时，假设$a_k \in A_{ij}$，则可用S_{ik}和 S_{kj}两子问题的最优解来构造S_{ij}的最优解，由16.2式：

$C[i,j] = C[i,k] + C[k,j] + 1, i+1 \leq k \leq j-1$， // k有$j-i-1$种选择

$|A_{ij}| = |A_{ik}| + |\{a_k\}| + |A_{kj}| = |A_{ik}| + |A_{kj}| + 1$

$$C[i,j] = \begin{cases} 0 & \text{当} S_{ij} = \emptyset \\ \max\limits_{i<k<j}\{C[i,j] + C[k,j] + 1\} & \text{当} S_{ij} \neq \emptyset \end{cases}$$

活动选择问题

动态规划到贪心算法的转化

可通过下面的定理简化其解，该定理也说明**贪心算法的正确性(*i.e.*，贪心选择的最优性)**。

Theorem. 1: 设S_{ij}是任一非空子问题，a_m是S_{ij}中具有**最早**完成时间的活动：$f_m = min\{f_k : a_k \in S_{ij}\}$，则：

1. 活动a_m必定被包含在S_{ij}的**某个**最优解中；
2. 子问题S_{im}是空集，使S_{mj}是唯一可能的非空集　//选a_m的目的

pf: 先证第2部分：

假定S_{im}非空，则$\exists a_k \in S_{im}$, 使$f_i \le s_k < f_k \le s_m$

$\because a_k$的完成时间先于a_m, 且$a_m \in S_{ij}$

这与a_m是S_{ij}的最早完成活动矛盾！

再证第1部分：

设A_{ij}是S_{ij}的某个最优解，假设A_{ij}中的活动已按完成时间单调递增排过序，a_k是其中第一个活动

若$a_k = a_m$，则问题已得证，即最优解包含a_m

若$a_k \neq a_m$，则构造子集$A'_{ij} = (A_{ij} - \{a_k\}) \cup \{a_m\}$，只需证$A'_{ij}$也是最优解即可

A_{ij}是S_{ij}的某个最优解

$\because a_k \in A_{ij} \subseteq S_{ij}$

$\therefore f_m \le f_k$

又$\because a_k$是A_{ij}中最早完成的任务，a_m比a_k更早完成

$\therefore |A'_{ij}| = |A_{ij}|$

$\therefore A'_{ij}$亦是S_{ij}的一个最优解，它包含a_m

活动选择问题

- 该定理的价值可简化问题的解
  - 在动态规划求解时，原问题S_{ij}可分解为两个子问题S_{ik}和S_{kj}求解，且这种分解有j-i-1中可能
  - 定理Theorem. 1将其简化为：
    - 求S_{ij}的最优解时只用到一个子问题，另一个子问题为空；
    - 只须考虑一种选择：即选S_{ij}中最早完成的活动
  - 该定理带来的另一个好处是：能以自顶向下的方式解每一个子问题

活动选择问题

- **对原问题$S = S_{0,n+1}$，选其中最早完成的活动a_{m1}**

  ∵ S中的活动已按完成时间单调增有序

  ∴ $m1 = 1$，下一子问题应是$S_{m1,\,n+1}$　　//已去掉于a_{m1}冲突的活动

- **选$S_{m1,\,n+1}$中最早完成的活动a_{m2}**

  ∵ 须将S中与a_{m1}冲突的活动删除才能得到$S_{m1,\,n+1}$

  ∴ m_2未必为2

- **一般形式，当选择a_{mi}加到解集中之后，需解的子问题变为$S_{mi,\,n+1}$**

显然，所选活动是按完成时间单调增有序的$(m1 < m2 < \cdots < mi < \cdots)$

活动选择问题

- 当某个a_{mi}加入解集合后，我们总是在**剩余**的活动中选择**第一个不与a_{mi}冲突**的活动加入解集合，该活动是能够**最早完成**且与a_{mi}**相容**的。
- 这种选择为剩余活动的调度留下了尽可能多的机会。即：留出尽可能多的时间给剩余的尚未调度的活动，以使解集合中包含的活动最多
- **每次选一个最早完成并与刚加入解集元素相容的活动**

活动选择问题

递归的贪心算法

- 输入：向量s和f（并假定已按f单调增有序）、子问题S_{ij}的下标
- 输出：S_{ij}的最优解
- 算法：

$RecursiveActivitySelector(s, f, i, j)\{$

// 求S_{ij}的最优解，a_i刚加入解集合，原问题的解：$i = 0,\ j = n + 1$

$m \leftarrow i + 1;$ //当$i = 0$时，有a_1必加入解集

$while(m < j)\ and\ (s_m < f_i)\ do$

// 将与a_i冲突的活动去掉，才能得到S_{ij}

$m \leftarrow m + 1;$ //在$a_{i+1}, a_{i+2}, \ldots, a_{j-1}$中找第1个与$a_i$ 相容的活动a_m

$if(m < j)\ then$ //$s_m \geq f_i,\ a_m$是S_{ij}中第1个活动，

//即在a_i完成后开始且最早能完成的活动

$return\ \{a_m\} \cup RecursiveActivitySelector(s, f, m, j);$

$else$

$return\ \emptyset;$ //a_j开始前能够完成的活动均与a_i冲突

活动选择问题

时间

- 若f要排序，则时间为$\Theta(nlgn)$
- 若f已排好序，则 $RecursiveActivetySelector(s, f, 0, n+1)$ 的时间为$\Theta(n)$

在所有的调用里，每个活动被检查一次，请注意每次循环检查时，活动序号只增加不减少，从$RAS(s, f, i, j)$ 调用到 $RAS(s, f, m, j)$时，必有 $i < m$。始终为$n+1$.

活动选择问题

迭代的贪心算法

上述递归很接近于尾递归，只是结束前多了一个Union操作，很容易转换为迭代算法：

- j 始终不变，$j = n + 1$
- 当一个活动a_i加入解集合之后，我们只要从a_i依次往后找到第一个不与a_i冲突的活动a_m，由于活动**事先按完成时间单调增有序**，故a_m是最早完成且与a_i相容的活动，它也是S_{ij}中的第一个活动

$GreadyActivitySelector(s, f)\{$ // f单调递增有序

$n \leftarrow length[s];$

$A \leftarrow \{a_1\};$ // A为解集合

$i \leftarrow 1;$ // i是最近加入解集合A的活动a_i的下标

$for\ m \leftarrow 2\ to\ n\ do$

$if(f_i \leq s_m)\ then\{$ // $i < m, a_m$与a_i相容

$A \leftarrow A \cup \{a_m\};$ // a_m是与a_i相容且完成时间最早的活动

$i \leftarrow m;$ // i仍记录最近加入A的活动a_i的下标

$\}$ // $endif$

$return\ A;$

$\}$

活动选择问题

注意，若直接给出迭代算法，一般要证明A确实为最优解。这一点由递归算法得以保证，亦可用归纳法直接证明：

- 总是存在一个最优解，第一步是贪心的选择，即选择活动a_1；
- 在已做的选择数目上做归纳法证明

活动选择问题

- 算法要点
  - **i 始终表示最近加入A的活动的下标**

    ∵活动已按完成时间有序

    ∴ f_i总是当前A中所有活动的最大完成时间：$f_i = max\{f_k : a_k \in A\}$

    当a_m加入A时，∵ $s_m \geq f_i$，∴ s_m也大于A中所有活动的完成时间，即a_m与A中所有活动相容
  - **时间：$O(n)$　　//f已排序的情况下**